

CLAIMS

What is claimed is:

1. A system to synchronize object management systems having a plurality of object management system components, comprising:

5 a distributed reader and writer's lock for each of the plurality of object management system components that communicates over a language interface and controls access to information shared by a corresponding object management system component within multiple object management systems;

a module that creates the distributed reader and writer's lock and serves as an agent for the object management system component; and

a list controller, which maintains a communications list and is adapted for communication with a communications controller, to which the distributed reader and writer's lock offloads management of a stub interface; wherein the distributed reader and writer's lock functions to:

request a local read lock and release a read lock;

request a local write lock and release a local write lock; and

request a remote write lock and release a remote write lock.

2. The system of claim 1, wherein each distributed reader and writer's lock communicates with corresponding locks on other object management systems through an ix_ring object that serves as a ring buffer.

3. The system of claim 1, wherein the distributed reader and writer's lock provides two callback functions while registering as a client of the list controller.

4. The system of claim 3, wherein one callback function is the creation and

initialization of an ix_base_t object as a stub interface to a new object management system being connected to the system.

5. The system of claim 4, wherein the language interface performs the initialization of the ix_base_t object generated stub initialization function.

5 6. The system of claim 3, wherein one callback function is the clean up and destruction of an ix_base_t object when an object management system is disconnected from the system.

7. The system of claim 6, wherein the language interface performs the clean up of the ix_base_t object generated stub clean up function.

8. The system of claim 1, wherein the module may have only one write lock at a time and several read locks.

9. The system of claim 8, wherein the write lock is granted to the module upon a release of all outstanding read locks and a grant and release of all outstanding read lock requests.

10. The system of claim 1, wherein the language interface uses an ix_base_t object to support a skeleton interface, which supports an incoming message, and an ix_base_t object to support a stub interface, which supports an outgoing message.

11. The system of claim 1, wherein the list controller provides a function to iterate through the distributed reader and writer's lock ix_base_t objects.

12. A method of requesting a local write lock, comprising:

20 waiting for a drop in pending status of a write lock, if a write request is pending;

incrementing a pending semaphore;

incrementing a local write lock count, if a local module has the write lock;

locking a lock contention mutex and creating a random number for resolving write lock contention;

checking a variable dedicated to write lock arbitration counting;

incrementing the arbitration count;

5 setting the write lock's arbitration identification and setting its priority by generating a bound random number;

releasing the lock contention mutex; and

making a write lock request.

13. The method of claim 12, wherein the write lock's arbitration identification is set to an identification of a local object management system.

14. The method of claim 12, wherein if the write lock request fails because of contention, the local lock write count is decremented.

15. A method of requesting a local read lock, comprising:

passing a local read lock request to a local lock component;

granting or blocking the local read lock request;

retrying if the local read lock request is blocked; and

repeating until a maximum number of retries is reached.

16. The method of claim 15, wherein the local read lock request is blocked if a remote module owns a write lock.

20 17. A method of requesting a write lock from a remote module, comprising:

locking a lock contention mutex and checking an arbitration count;

resolving contention for a write lock or incrementing the arbitration count; and

releasing the lock contention mutex or forwarding the request to a lock component.

18. The method of claim 17, wherein if the arbitration count is not greater than zero, the arbitration count is incremented.

19. The method of claim 17, wherein an arbiter examines a priority value and identification value.

20. The method of claim 19, wherein if a requested priority differs from a current priority, an arbitration winner is decided by the priority value.

21. A method of releasing a local write lock, comprising:

setting a clear pending lock;

locking a write lock contention mutex and decrementing a write lock count;

clearing write lock contention variables, if the write lock count is zero;

releasing the lock contention mutex;

calling a write lock release function; and

releasing the clear pending lock.

22. The method of claim 21, wherein the clear pending lock is set to prevent a local object management system from processing another request until the lock is released from remote object management systems.

23. The method of claim 21, wherein calling a write lock release function occurs in two stages of clearing arbitration variables and enabling arbitration for a following request.

24. The method of claim 21, wherein a write lock count function and arbitration count function are combined into one variable.

25. A method of releasing a remote write lock, comprising:

setting a clear pending lock;

locking a write lock contention mutex and clearing write lock contention variables;

clearing a writer's lock from a local lock component;

releasing the write lock contention mutex; and

clearing the clear pending lock.

26. The method of claim 25, wherein the write lock contention variables are

5 identification and priority.

27. A system for a local write lock request comprising a computer readable medium

and a computer readable program code stored on the computer readable medium having

instructions to:

wait for a drop in pending status of a write lock, if a write request is pending;

augment a pending semaphore;

augment a local write lock count, if a local module has the write lock;

lock a lock contention mutex and create a random number for resolving write lock contention;

check a variable dedicated to write lock arbitration counting;

augment the arbitration count;

set the write lock's arbitration identification and set the write lock's priority by

generating a bound random number;

release the lock contention mutex; and

request a write lock.

28. The system of claim 27, wherein the write lock's arbitration identification is set to

an identification of a local object management system.

29. A system for a local read lock request comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:

pass a local read lock request to a local lock component;

award or deny the local read lock request;

retry if the local read lock request is denied; and

repeat until a maximum number of retries is reached.

30. The system of claim 29, wherein the local read lock request is denied if a write lock is owned by a remote module.

31. A system for a remote write lock request comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:

lock a lock contention mutex and check an arbitration count;

resolve contention for a write lock if the arbitration count is greater than zero, or augment the arbitration count if the arbitration count is not greater than zero; and

release the lock contention mutex or forward the request to a lock component.

32. The system of claim 31, wherein the instructions are adapted to provide to an arbiter to examine a priority value and identification value.

33. A system for a local write lock release comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:

set a clear pending lock;

lock a write lock contention mutex and decrement a write lock count;

clear write lock contention variables, if the write lock count is zero;
 release the lock contention mutex;
 call a write lock release function; and
 release the clear pending lock.

5 34. The system of claim 33, wherein the clear pending lock is set to prevent a local
 object management system from processing another request until the lock is released from
 remote object management systems.

35. A system for a remote write lock release comprising a computer readable medium
 and a computer readable program code stored on the computer readable medium having
 instructions to:

set a clear pending lock;
 lock a write lock contention mutex and clear write lock contention variables;
 clear a writer's lock from a local lock component;
 release the write lock contention mutex; and
 clear the clear pending lock.

36. The system of claim 35, wherein the write lock contention variables are
 identification and priority.

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995